DOCUMENT RESUME

ED 074 765 EM 010 963

AUTHOR	Henson, Jane C.; Manry, William F., IV
TITLE	APL: An Intro.
INSTITUTION	Atlanta Public Schools, Ga.
PUB DATE	71
NOTE	39p.
AVAILABLE FROM	Terminal Services Coordinator, Information Processing System, Atlanta Public Schools, 218 Pryor St., S.W., Atlanta, Georgia 30303 ($2.00)
EDRS PRICE	MF-$0.65 HC Not Available from EDRS.
DESCRIPTORS	*Computer Assisted Instruction; *Computer Oriented Programs; *Manuals; *Programing Languages; Secondary Education
IDENTIFIERS	APL; *A Programing Language

ABSTRACT

This guide, written for high school students in Atlanta, Ga., explains how to use the computer system available to Atlanta high school students and teachers. The system is a remote terminal system. The language used for it is called APL/360 (A Programming Language for the IBM 360 computer). The guide explains how to sign on the computer, send messages between users, and write elementary programs. A glossary of programing symbols for APL is included. (JK)

APL

an intro

IMPORTANT NUMBERS

FILL IN THESE BLANKS WITH THE CORRECT NUMBERS, WHICH YOU CAN OBTAIN FROM THE APL COORDINATOR AT YOUR SCHOOL:

MY APL USER NUMBER:

APL SYSTEM TELEPHONE NUMBER:

TERMINAL SYSTEMS COORDINATOR TELEPHONE NUMBER:

IF THIS BOOK IS FOUND, PLEASE RETURN TO:

NAME:

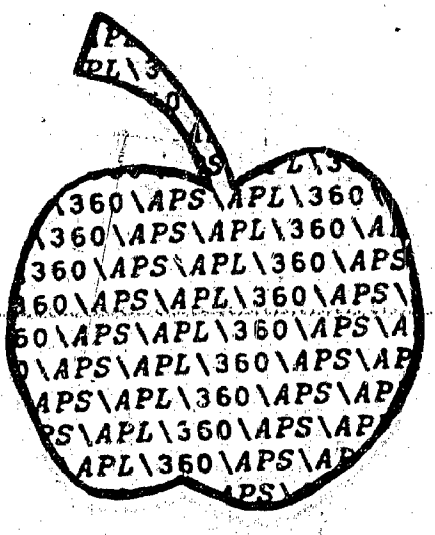
GRADE OR CLASS:

HOME ADDRESS:

HOME TELEPHONE NUMBER:

APL: an intro

by Jane C. Henson and William F. Manry, IV



Published by

Atlanta Public Schools

John W. Letson
Superintendent

Atlanta, Georgia
1971

This manual is dedicated to the students and faculty of the Atlanta Public Schools. Its purpose is to introduce the use of APL/360, but it should also interest the experienced user. APL/360 is modular in design, so you will find that you need learn only that portion of the language required for a particular application.

The uses of APL/360 are limited only by your imagination, as you will soon learn. Your mastery of the language will determine your success in the use of this computer system.

We hope you will benefit from this book, and we invite your constructive criticism.

Thomas J. McConnell, Jr.
Director, Information Processing System

* * * * *

The authors would like to extend their thanks to all those who helped in the production of this book. Our special thanks go to Holman Mayfield and Fran McLaughlin of the Publications Department, and to Bernard A. McIlhany, the Terminal Services Coordinator, who prepared the final copy for printing. We received help from the Datel Corporation, the Data Division of the Southern Bell Telephone Company, and the IBM Corporation. Thanks, also, to those of the Instruction Division who offered many helpful suggestions. Finally, our appreciation to all the members of the Operations Department of the Information Processing System, without whose constant efforts API/360 in the Atlanta Public Schools would not be possible.

The Authors

* * * * *

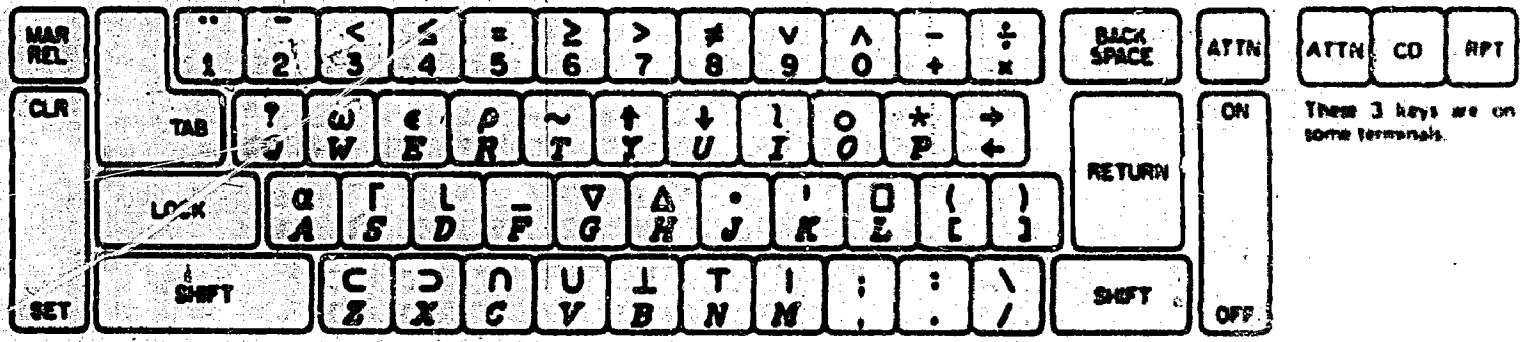
SECOND EDITION. Revised April, 1971.
(Printing indicated by last digit.) 10 9 8 7 6 5 4 3 2 1

Additional copies may be obtained from: Terminal Services Coordinator, Information Processing System, Atlanta Public Schools, 218 Pryor St.S.W., Atlanta, Georgia 30303. Price: $2.00.

TABLE OF CONTENTS

		PAGE
I.	Introduction	
	A. Remote Terminal	1
	B. APL - What It Is	1
	C. Keyboard	1
	D. Sign-on	1
	E. Sign-off	3
II.	Communication	
	A. Messages Between Users	3
	B. Ports	3
III.	The APL Language	
	A. Input/Output	3
	B. Basic Functions	4
	C. Order of Execution: What Happens When	7
	D. Logic	11
	E. Sets of Numbers or Variables	12
	F. Error Messages	14
IV.	Programs	
	A. Writing Programs	15
V.	More About the APL System	
	A. Libraries and User Account Numbers	22
	B. Workspaces	23
	C. Public Libraries	24
TABLES:	Error Reports	26
	System Commands	26
	I-beam functions	27
	Symbol Chart	28
	Overstrikes	30
	Mysterii	31

Where to Direct Comments and Suggestions.....INSIDE BACK COVER



APL\360 KEYBOARD

FIGURE 1.

APL - WHAT IT IS

Right now, a computer system is available for use by Atlanta high school students (and teachers). If your school is equipped with this facility, ask your APL-Coordinator for permission to use it, and give it a try. This book is designed to explain some basic facts about using the computer. It will show you how to get started, and you can take it from there. You need not have had any special or advanced mathematics instruction -- basic math is plenty. If you have had Algebra or Geometry, you're that much better off. However, no matter what your math instruction, you can actually teach yourself some real computer programming with only a little practice.

The system used by the Atlanta schools is called a remote terminal system. This means that your connection with the actual computer (located in downtown Atlanta) is made from some distance away with a terminal, which looks like a large typewriter. Standard phone lines carry the connection -- in fact, you will use a telephone, or Dataset, to "call" the computer and connect the terminal.

Because the computer doesn't understand English as you know it, you must use a special language. The particular language you will use is called *APL\360*. The *APL* stands for A Programming Language; the 360 for the type of computer you will be connected with: an IBM System/360. You will find that APL is similar in many ways to your regular arithmetic. the symbol for addition is +; for division, ÷ etc. However, it is also different in some ways.

The APL keyboard (figure 1) is designed like a regular typewriter keyboard. Notice that the normal alphabet keys type letters in unshifted (lower case) position only, and these letters are italicized capitals: *ABCDEFGHIJKLMNOPQRSTUVWXYZ*. When shifted (upper case), the keys type only symbols or punctuation. The number keys are in the same position as a regular typewriter, but the numeral 1 is distinguished from the letter *I* or *L* .

Please notice especially the keys on the right marked ATTN (attention) and RETURN. Also, the ON/OFF switch is located below the ATTN key. You will use these three keys often.

SIGN-ON

In order to make the connection between your terminal and the computer you must go through a procedure called SIGN-ON. To perform a sign-on and begin a session with the computer there are certain steps you must follow. Before trying a sign-on ask your APL-Coordinator for a sign-on ACCOUNT NUMBER. Read the instructions below once completely so that you will have some idea of what to do. You don't have to memorize the instructions -- just be familiar with them. Follow them exactly for your first few sign-on's, at least until you get the hang of it. If you make a mistake, don't worry! You can't hurt APL. Simply start over and try again. (See p.25)

1. Insert paper into the terminal. Use whatever paper your APL-Coordinator provides.
2. Sit comfortably at the terminal with the dataset (or telephone) within reach.
3. Turn the terminal's ON/OFF switch to ON. The switch is located on the right of the keyboard.

4. Look over on the left side of the terminal (not the typewriter part -- the desk or table part) and find the switch marked LCL/COM. Be sure it is switched to COM. Some terminals say LOC instead of LCL. In local (LCL or LOC) your terminal is a regular typewriter!
5. Pick up the telephone receiver on the dataset and press the button marked TALK. It will stay down and you will hear a dial tone.
6. Dial the computer number: _____. If you hear a busy-signal or a click and no other response, either the telephone lines are full or the computer is DOWN (not available). If you hear a ringing and no answer, again the computer is DOWN. You should try again later.
7. When the computer answers properly, you will hear a high, piercing tone in the receiver. When this happens, press the DATA button on top of the dataset firmly and it will light. You may then hang up. If the data button doesn't light at all, check your telephone connections and the power line to the dataset.
8. When the DATA button has been pressed, the keyboard on the terminal will unlock, permitting you to type your sign-on. To do this, type a) -- right parenthesis -- followed by your account number. (Do not put a space between the right parenthesis and the number, or between the digits of the number!) When through typing, press the RETURN key. This will move the type element back to the margin, will advance the paper one line, and at the same time will send the line you just typed to the computer.

```
)99999
```

9. If the computer's response is *INCORRECT SIGN-ON*, check to be sure you typed correctly. The computer will hold the connection about 10 seconds for you to try again before you must dial the number again. If the response is either *NUMBER IN USE* or *NUMBER NOT IN SYSTEM*, ask your APL-Coordinator for assistance. If the computer receives your sign-on properly, it will type out some information about the time and the date of the sign-on, and below that it will type the name of the computer system. Note the example below:

```
)99999
003) 10.41.30 05/07/70 ALGEBRA

A P L \ 3 6 0 EXPRESS
```

The information on the second line above will vary depending on the time, date, and the sign-on account number used. The word *EXPRESS* on the bottom line means that the amount of time you are connected is automatically limited by the computer. This usually varies from 15 minutes to one hour. If you don't sign off before your time is up, you will automatically be signed off at the end of the limit.

When all the above has been accomplished you are ready to go to work on APL. However, for future reference, here is the procedure you must follow to sign off:

1. Type)OFF and press the RETURN key. Wait until the computer prints some information about the time you were connected, etc.
2. When the computer stops printing, turn the terminal ON/OFF switch to OFF.

At this point there is something that should be stressed for all beginning users: the APL terminal system is ideal for learning because there is no way that you can damage anything by typing on the terminal. The worst that can happen is a simple error, which has no effect on either terminal or computer. For these reasons, we urge students to go ahead and "experiment" -- try some new things. It's the best way to learn!

MESSAGES BETWEEN USERS

If something goes wrong which you do not understand you can send a message to the operator terminal downtown:

```
you:    )OPRN ANY 1 LINE MSG      (leave one space between
APL:SENT                           the command and your message)
```

When you hit the RETURN key the computer responds after a slight pause with *SENT* , meaning the message has been typed out on the operator terminal. Now continue with your work.

You can also learn who else is using APL by typing the system command:

```
you:          )PORTS
APL:  OPR OPE  <----------  (this is the operator)
      001 WGA
      003 UNC
      005 CAT
```

To send a message to someone, first find out their port number (it changes each time they sign on) then type)MSGN a space, the port number, another space, and then the message. Once you hit RETURN the message is transmitted.

```
you:          )MSGN 3 IS MR. JONES THERE?
APL:    SENT
```

Again, the response is *SENT* although it may be a while before *SENT* is printed. If you get impatient and wish to continue working, you may hit ATTN, but if you do, the reply *MESSAGE LOST* will be received. The ATTENTION key has interrupted the message carrier and the message will not be received by anyone.

A copy of all messages between users is printed on a monitor printer at the computer center. Transmission of obscene messages via teleprocessing systems is punishable under Federal and State laws by a fine or a prison sentence, or both.

INPUT/OUTPUT

When communicating with the computer , you must be able to distinguish who typed what. Normally, everything you type (except literal input) is indented six spaces. Everything the computer types

is at the left margin. The computer can handle letters as well as numbers. Letters, or alphabetic data, are all capital stating.

The APL language, like all computer languages, requires precision in instructions. The computer will execute exactly what you type; you must be extremely careful if you are to get the computer to do what you want. If you are not precise and type in something incorrectly, the computer will either execute it incorrectly or tell you that you have made an error. Either way, you have not damaged the computer. Simply try your command again. APL can handle variables such as X, Y, and I. To assign a variable a value use the left arrow (←) after the variable name, then assign the value.

X←3 replaces every X variable with 3. Variables can be named anything less than 77 characters long. Every variable name must begin with a letter; after that any alphabetic or numeric characters are okay (do not include spaces).

[illegible]←7

is an acceptable variable with the name [illegible] and with the value of 7.

APL is fun to learn. Do not worry if you make mistakes; you will not break the computer. Your command of the language will improve with experience.

BASIC FUNCTIONS

APL and arithmetic use the same symbols for addition, subtraction, multiplication, and division. The keys for these are in the upper right corner of the keyboard. An upshift with the - key (hold the shift key while hitting the - key) prints a subtraction sign. The upshift of the ÷ key prints a division sign.

To add 4 and 7, type a 4, a + and a 7; then press the RETURN key on the right of the keyboard.

```
you:    4+7        (hit RETURN)
APL:11
```

Spaces within the problem are not necessary but harmless if inserted.

Try several problems with each operation.

```
you:    4-7        (hit RETURN)
APL:¯3
```

Notice that the negative sign in front of the three is different from the minus sign. The negative sign indicates a number less than zero; the minus sign indicates that a subtraction operation is to be performed.

```
you:    7-4        (hit RETURN)
APL:3

you:    4÷2        (hit RETURN)
APL:2
```

```
you:    [illegible]        (hit RETURN)
APL: [illegible]

you:    [illegible]        (the negative sign is made
APL: ¯[illegible]             with an uptick ¯)

you:    [illegible]
APL: [illegible]
```

In arithmetic, you probably are used to leaving out the multiplication sign. APL requires every sign for every operation to be included.

```
you:    [illegible]        (hit RETURN)
APL: SYNTAX ERROR          (the computer doesn't know
        [illegible]           what to do)
        ^
```

Did you want to multiply [illegible] times [illegible]? Try this instead:

```
you:    [illegible]        (hit RETURN)
APL: ¯[illegible]

you:    [illegible]        (hit RETURN)
APL: [illegible]
```

Now try using variables to perform operations. Every variable must have a numeric value.

```
you:    A←7        (hit RETURN)
```

You have assigned the value 7 to the variable A. (The computer makes no response because no operation has been requested.) Now, anytime you use or ask for A it will have the value of 7.

```
you:    4+A        (hit RETURN)
APL: 11

you:    [illegible]        (hit RETURN)
APL: [illegible]

you:    B←10       (hit RETURN)
you:    A+B        (hit RETURN)
APL: 17
```

In addition to operations with two numbers, operations can work on only one number. An operation sign in front of a number or variable causes certain operations to be performed on that number or variable. A plus sign in front of a single value gives the value itself.

```
you:    +7         (hit RETURN)
APL: 7

you:    A←¯3       (hit RETURN)
you:    +A         (hit RETURN)
APL: ¯3
```

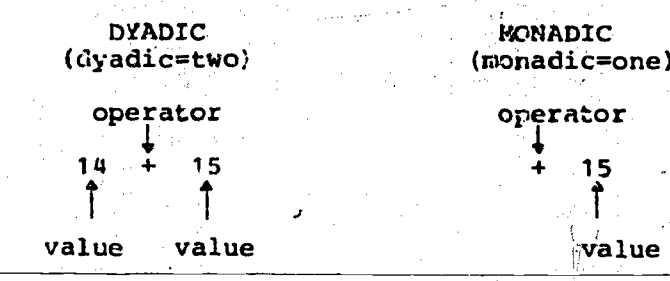
The subtraction sign in front of a value gives the opposite of the number.

```
you:    -5          (hit RETURN)
APL:¯5
```

Dyadic operations involve an operator with values on either side of the operator. Dyadic use of + is, for example, 4+7. The + has a value before and after it. Monadic operation involves an operator with a value only to its right. Monadic use of + is +7. The + has nothing on its left.

Monadic use of ÷ gives the reciprocal of a number. The reciprocal of 17 is 1/17 (APL does not give fractions, but instead, gives the decimal equivalent of 1/17).

```
you:    ÷17         (hit RETURN)
APL:0.05882352941
```



There is also a monadic use for ×, which you may learn from your APL manual.

To write seven squared (7^2) in APL, write 7*2 rather than trying to type the raised number. APL has no superscript numbers to use as exponents. The asterisk, or star, is the power or exponent sign.

```
you:    7*2         (hit RETURN)
APL:49
```

Seven times seven (or seven squared) is forty-nine.

```
you:    3*3
APL:27
```

APL has no specific function to find the square root of a number. However, the square root of 4 is the same as raising 4 to the one-half power, or, in APL, 4*0.5.

```
you:    4*.5
APL:2
```

The square root of four is two. APL will also do other roots such as cube root, and fifth root; all you have to do is type, instead of .5, the decimal equivalent of the reciprocal of the root number.

```
you:    8*.333333333
APL:2
```

Combining several operations into one step includes ORDER OF EXECUTION. In arithmetic the exponents are evaluated first, then the multiplication and division, and finally, addition and subtraction.

ORDER OF EXECUTION: WHAT HAPPENS WHEN

APL does not follow this order. Regardless of operation, APL executes from right to left.

ORDINARY ARITHMETIC EXECUTION

3x4+7

multiplication first: [3x4] +7

replacement: [12] +7

addition last: [12+7]

[19]

APL EXECUTION; RIGHT-TO-LEFT

3×4+7

right operation first: 3× [4+7]

replacement: 3× [11]

next operation to the left: [3×11]

replacement and results: [33]

So if you forget the APL right-to-left rule, you could get a wrong answer. Look at this example:

ORDINARY ARITHMETIC EXECUTION

3÷1+2×1×2

division first: [3÷1] +2×1×2

replacement: [3] +2×1×2

multiplication next: 3+ [2×1] ×2

replacement: 3+ [2] ×2

multiplication repeated: 3+ [2×2]

replacement: 3+ [4]

addition last: [3+4]

replacement and results: [7]

APL EXECUTION: TO-LEFT

```
                    3÷1+2×1×2

right operation
first               3÷1+2× [1×2]
                           ↓
replacement         3÷1+2× [2]

next operation
to the left         3÷1+ [2×2]
                         ↓
replacement         3÷1+ [4]

next operation
to the left         3÷ [1+4]
                       ↓
replacement         3÷ [5]

next operation      [3÷5]
                       ↓
replacement         [0.6]
and results
```

Interesting, isn't it? Well, there is a way to get the right answer: you may change order of execution by using parentheses. Whenever ( ) appear in a line of APL, whatever is within the parentheses is executed first. (Execution inside or outside of parentheses always follows the right to left rule.) The final value found within the parentheses replaces the parentheses. Once all parentheses have been evaluated, execution continues right to left.

3÷1+2×1×2 results in 7 when using ordinary arithmetic execution; .6 when using APL execution. To make the results 7 when using APL execution, use parentheses.

```
(3÷1)+2×1×2
[(3÷1)] +2×1×2    execute within parentheses
   ↓
[3] +2×1×2        replacement

 +2× [1×2]        after ( ) evaluation,
      ↓           execution continues with
                  right operation
 +2× [2]          replacement

3+ [2×2]          next operation
    ↓              to the left
3+ [4]            replacement

[3+4]             next operation
   ↓
[7]               results
```

"Factorial" is a function used in Algebra. The factorial of five is the same as 5x4x3x2x1 [illegible]e factorial of 3 is 3x2x1. In APL, the factorial of [illegible]. The ! is made with a ' (upshift K), a backspace, a[illegible] decim[illegible] point. This is called "overstrike". Several other APL characters are also made by overstrikes.

```
you:   !3  (hit RETURN)  (3x2x1)
APL:6


you:   !5   (hit RETURN)
APL:120
```

To perform operations involving circles, a value for pi is needed. The formula for the area of a circle is πr^2 . To find the area of a circle with a radius of 3 inches: $\pi 3^2$. Pi is approximately 3.14159265 and is unending. In APL, pi is provided by the monadic symbol ○ (upshift letter *O*), read "pi times". Be careful not to confuse ○ with the upshift *J*, the letter *O*, or the numeral zero.

○	(upshift *O*)	called "circle" or "pi times"
∘	(upshift *J*)	called "small circle"
O		letter O
0		number zero

```
you:    ○1  (hit RETURN)
APL: 3.141592654
```

No × sign is needed because monadic ○ means pi times. This is an exception where the multiplication operator is not required, and multiplication is implied.

To find the area of a circle with radius 3:

```
you:    ○3*2
APL:  28.27433388
```

Since pi is a never-ending decimal, the computer approximates pi to ten digits. Therefore, the answer involving ○ is a number rather than a multiple of pi.

To find the circumference of a circle (circumference= πd) with diameter 6:

```
you:    ○6
APL: 18.84955592
```

You may use APL to select positive whole numbers at random. By typing ?10 you are asking the computer to select any number between (and including) 1 and 10. It will respond with one random number. The ? is read "roll".

```
you:    ?10
APL: 8
```

The computer randomly selected the number 8 from 1 2 3 4 5 6 7 8 9 10.

```
      you:      ?10
      APL: 2
```

This time, the computer randomly selected the number 2.

```
      you:      ?400
      APL: 293
```

APL provides several unique operations. They were created because the operations they perform are often useful in various programs.

⌈6÷4 takes the next whole number greater than or equal to six divided by four. In other words, ⌈ rounds up to the next whole number.

```
      you:      ⌈6÷4
      APL: 2
```

Six divided by four is one and one-half. When you round up this number , the next largest whole number is two.

```
      ⌈1.237
2
      ⌈¯.432
0
```

⌈ takes the "ceiling" of a number and is employed monadically.

The opposite of ⌈ is ⌊ which takes the "floor" of a number, the next whole number less than or equal to the number to the right of ⌊.

```
      you:      ⌊3.764
      APL: 3
      ⌊3
3
      ⌊○1
3
```

Here you are rounding down numbers. ○1 is an approximation of pi: 3.141592654. When you take the floor of pi you come up with 3.

```
      -⌊¯3.245
-4
```

Dyadically, ⌈ takes the maximum of the two numbers on either side of it. ⌊ takes the minimum.

```
      4⌈7
7
```

Seven is greater than four, therefore the "ceiling" choses seven.

```
 7⌈4
7

 4⌊7
4

 ¯10.4 ⌊ ¯2.579
¯10.4

 A←33
 B←¯19
 A⌈B
33
```

Residue, an upright bar | (upshift *M*), takes the remainder of a division problem.

```
      26
4 ) 105
      8
     25
     24
      1 ←  (The remainder is one).
```

```
you:      4|105
APL:  1
```

To get the total answer and the remainder printed out:

```
you:      ⌊105÷4
APL:  26

you:      4|105
APL:   1
```

LOGIC

In APL there are ways of comparing numbers (different from operating on them, such as by addition, subtraction, etc.). For instance, if you wanted to know if variable *A* was larger than or equal to variable *B*, you could do this:

```
      A←5

      B←4+1

      A=B

1

      A<B

0

      A≥B

1
```

In each case the result was either a one or a zero. The signs < ≤ = ≠ ≥ > are called logical operators (all have the same meaning as in regular math). If the statement using them is true, the result is 1. If it is false, the result is 0. One and zero are the only possible answers you can get from using the logical operators. Notice the following examples:

```
      A←23
      C←0
      B←¯10
      A>A+B
1
      5×(A=A)
5
      A=A
1
```

In the next to the last example, the response was 5, not a one or zero. This is because you multiplied 5 times the result of A=A. Since A=A produces a one, then you really said 5×1, or 5.

SETS OF NUMBERS OR VARIABLES

Assume you want to add the number 2 to several numbers, namely 3, 5, 7, 9, and 11. You can type in

```
      2+3
5
      2+5
7
      2+7
9
      2+9
11
      2+11
13
```

or you may do it all in one line and save a lot of effort.

```
      2+3 5 7 9 11
5 7 9 11 13
```

This is addition using a vector. A vector is a series of numbers.

WHEN TYPING A VECTOR, ALWAYS PUT AT LEAST ONE SPACE BETWEEN EACH ELEMENT.

When operating with vectors the computer responds with another vector.

The last number is 5 because 6+¯1 (¯1 is negative one, remember?) is 5. Or, if you wish, you may put the vector on the left of the operator.

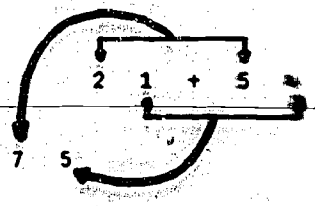
```
      3 2 1 0 ¯1+6
5 8 7 6 5

      3 4 5 6÷4    (divide each number
0.75 1 1.25 1.5            by 4)
```

Here is another example:

```
      1 2 3 4*2
1 4 9 16
```

When a vector appears on both sides of an operation sign, the operation applies to both vectors, element by element.



The first element of the left vector (2) is added to the first element of the right vector (5) for the first element of the resultant vector (7). The second element of the left vector (1) is added to the second element of the right vector(4) for the second element of the resultant vector (5). This applies to vectors of any length. However both vectors must be of the same length. If you try to operate on vectors of different lengths, the computer will tell you

```
LENGTH ERROR
```

If you get this response, display each vector to see what its contents are. Reassign one vector to conform in length to the other vector.

```
      A←C
LENGTH ERROR
      A
1 2 3 4
      C
1 2 3
```

C contains only 3 numbers while *A* contains 4 numbers. It is impossible for the computer to operate on these two unequal length vectors. You must change the number of values in one of the vectors so that the length of one will equal the length of the other. Then the computer can operate on them.

```
      C←1 2 3 4
      A+C
2 4 6 8
```

APL is not confined to working with numbers. It can also work with letters, or alphabetic characters, which are not variables. These are called literals.

```
      A←'FISHERMAN'
      A
FISHERMAN

      C←'FOUR'
      C
FOUR
```

If more than one character is within the quotes, it becomes a literal vector. Both of these are examples of literal vectors. *A* was assigned the value of everything within the single quotes, *FISHERMAN*. Every literal character must be put within single quotes.

To index a literal vector (such as *A* and *C*), in other words, to rearrange selected letters from their contents, type the vector name (*A*), a [(left bracket), then the number of the position of the first letter, a blank, then the second position number, and so on until the selection is complete; then type] and return:

```
      A[3 5 5 7]
SEEM
      A[1 8 7 5]
FAME
```

ERROR MESSAGES

By now, if you've had much time to use the computer terminal, you have probably gotten at least several different kinds of ERROR REPORTS (in fact, if you've been trying real hard you may have gotten seven or eight!). This is one of the great advantages of APL: if an error is made, the computer tells you what kind of error, not just the fact that it occurred. Right now, a knowledge of some of the simple error reports may be of some help in your use of APL. The APL notation below is exactly what the computer types out when the particular error occurs.

VALUE ERROR

This happens when you try to use a variable with no value assigned to it. For instance, if you type *A←5* and then type *A+B*, you will get a value error with the little marker (^) showing the beginning letter in the variable without a value.

```
      A←5
      A+B
VALUE ERROR
      A+B
        ^
```

SYNTAX ERROR

When the report Syntax Error is received, it indicates that the command was poorly or incorrectly made. For example, if you had an extra parenthesis in a formula, the computer replies with a Syntax Error.

```
      ((2*3)-4.5
SYNTAX ERROR
      ((2*3)-4.5
      ^
```

DOMAIN ERROR

If you get a Domain Error , then you have tried to perform an operation that can't take place with the numbers you provided. You know that it is impossible to take the square root of a negative number (ignoring imaginary numbers). If you try it in APL, the computer tells you that it is impossible to do.

```
      ¯9*0.5
DOMAIN ERROR
      ¯9*0.5
        ^
```

WRITING PROGRAMS

Programs are written to teach the computer to do something it doesn't already know. Programs set up steps; the computer executes them exactly as they are written, line by line. Programs are useful when a process is to be used over and over again.

APL can be in one of two states or modes. It is always in one mode, and it can never be in both modes at the same time. The execution mode enables the computer to immediately carry out instructions you type in when you hit the RETURN key. The computer gives an immediate response.

The execution mode can be changed to the definition mode by typing a ∇ (called del, upshift "G"), followed by a name for the program you are creating:

```
      ∇ANYNAME
```

In this state, the computer is able to receive instructions (as many lines as you like) without executing any of them. To return to execution mode after the program is complete, type another ∇ (del). The program of instructions entered by the user during definition

mode can only be carried out in execution mode.

PROGRAM: Converting degrees in Centigrade to degrees in Fahrenheit.

Use the formula:

C=5/9 (F-32)

To begin, the formula must be written in APL.

CONVENTIONAL	APL
F-32	F-32
5/9	5÷9
C	C
C=5/9 (f-32)	C←5÷9×F-32

Now check carefully the order of execution

5÷9×F-32
5÷9× F-32
5÷ 9×(F-32) next operation to the left.

This is not right. F-32 should be multiplied by 5÷9 , not just nine. Overiding the ordinary order of execution is necessary.

C←(5÷9)×F-32

Check order of execution again.

(5÷9)×F-32
5/9 ×F-32
5/9 × F-32
5/9× VALUE
5/9×VALUE
ANSWER

This is the desired result. The formula is now in APL.

Here is how to put it into a program:

You type a ∇ and the name of the program (no spaces).

∇DEGREES

When you hit RETURN, indicating that you are finished with that line, the computer responds with [1] meaning that it is ready to receive the first line of instructions. APL automatically numbers your program lines for you.

∇DEGREES

Now type in the formula, using APL notation.

```
      [1]   C←(5÷9)×F-32
```

When you hit RETURN, you are finished with line one. The computer types [2] and waits for your instructions.

If at any time you make a typing error and realize it before you hit RETURN, backspace to the error, hit ATTN and retype the rest the line.

```
      [5]   A←B×C
                ^
                ←C
```

If you realize your mistake after you have hit RETURN, on the next line, instead of typing what goes there type [then the number of the line which has errors, type a] and RETURN; the line is now ready to be retyped completely, like this:

```
      [2]   [1]
```

You want the program next to print the evaluation for *C* so for line 2 type *C* and RETURN.

```
          ∇DEGREES
      [1] C←(5÷9)×F-32
      [2] C
      [3]
```

You are finished with the program, and want to return to execution mode. Type ∇, and RETURN. In other words,∇ begins and ends definition mode. (The computer will not type anything in response.) Here is how your program looks now:

```
          ∇DEGREES
      [1] C←(5÷9)×F-32
      [2] C
      [3] ∇
```

To make the program run, first assign a value for *F*, your variable (*C* does not need to be assigned a value because it is given a value by the program).

```
          F←32    (hit RETURN)
```

Then, type the name of the program:

```
          DEGREES
```

When you hit RETURN, the program will execute using the assigned value for *F* and will print the evaluated value for *C*.

Here are some examples:

1.
```
      F←32
          DEGREES
0
```

```
      2.   F←100 ⍋
           DEGREES
           37.77777778
```

Now, here is how you can convert several values at one time:

```
           F←0 32 100 212
           DEGREES
           17.77777778 0 37.77777778 100
```

If you use a vector for several values for *F*, you will get several computed values for *C* !

Here is another program example: Converting feet to yards.

Let X be the number of feet, which you enter as a variable. X divided by three (3 feet in a yard) will be the number of yards. If 3 does not divide X evenly, the result will be a decimal. Suppose you want the remainder left in feet. How would you write the program to print the number of yards and the number of feet left over?

To remove the decimal from X/3 use ⌊. To print the remainder after division, use residue: 3|*X* .

Now you are ready to tell the computer your program:

```
           ∇YARDS
           [1] Y←⌊X÷3    assigns to Y the number of yards
           [2] F←3|X     assigns to F the number of feet
           [3] Y           print Y
           [4] F           print F
           [5] ∇
```

To run the program, assign a value for *X* and then type the program name.

```
           X←11
           YARDS
           3
           2
```

11 feet equals 3 yards, 2 feet.

IF YOU DO NOT FULLY UNDERSTAND, GO BACK TO PAGE 15 AND REVIEW THIS SECTION SO FAR.

The computer can print words if you tell it to. You type exactly what you want printed (begun and ended by a single quote mark or apostrophe, which is the upshift *K* key).

You can have "yards" and "feet" printed in the output; rewrite the program. To do this you must title the new program something different.

```
           ∇YARDS1
           [1] Y←⌊X÷3
           [2] B←'YARDS'
           [3] F←3|X
```

```
[4] A←'FEET'
[5] F;B;F;A
[6] ∇
```

Line 5 prints out the assigned values for these variables. They are separated by : because Y and Y are numeric characters while A and A are alphabetic characters. Output for different types of characters must be separated by a semicolon. Now, run your new program:

```
X←11
YARDS:
3:YARDS:FEET
```

This is a little hard to read. If you want to put spaces in the output for easier reading, the spaces must be placed within the quotes in lines 2 and 6. Line 2 should read:

```
[2]   B←' YARDS '
```

And line 6 should read:

```
[6]   A←' FEET'
```

Now run the program again.

```
X←11
YARDS:
3 YARDS 2 FEET
```

Here is another sample program: To average numbers, add them up and divide the total by the number of numbers. Translate this to APL:

```
SUM←A+B+C+D
SUM:4
```

To execute this program, a value for each variable must be entered. In this example, you can only average the four numbers A, B, C, and D. Could you use vectors to handle more numbers? The limited ability of this program makes it not worth writing. Vectors will be ideal for this problem.

Let X be a vector of numbers which you wish to average. Now the problem is adding them up. APL very conveniently has a / operator (right slash or reduction) designed especially for vectors. The operation / reduces a vector to one number by inserting the operator that precedes / between each element of the vector.

```
+/1 2 3 4
means 1+2+3+4
which is 10
```

+/X will add each element of vector X , regardless of the length of X.

Now that this is worked out, you have to figure a way to get the computer to count the number of numbers in the vector so that the division can be done.

APL has an operator ρ (the Greek letter rho) to determine the length of a vector. ρX will yield the length of X , or the number of numbers in the vector X.

```
      Y←1 3 5 7 9 0 8
      ρY
7
```

There are seven numbers in the vector Y .

Write the program:

```
      ∇AVG
[1]  Y←(+/X)÷ρX
[2]  Y
[3]  ∇


      X←19 20 35 67
      AVG
35.25
```

Assume you have another vector of integers, Y. You do not want to average it by itself but you want to find the average of all elements of X plus all elements of Y. You could reenter a different value for X , if you wish, but if the vectors are long, typing in all these numbers is a lot of trouble. APL has an easy way to do this called catenation. Catenating vectors with a , (comma) puts the vectors together into one single, longer vector.

```
      Y← 17 29 30
      Z←X,Y
      Z      (display Z )
19 20 35 67 17 29 30
```

Now use your AVG program to average this longer vector.

```
      X←Z
      AVG

31
```

PROGRAM: Counting by two's to a given number.

You count by two's by adding two to each previous number starting with zero. To write a program in APL this will involve loop.

Begin with zero :

```
      X←0
```

Add two to that and reassign X

```
      X←X+2
```

then when it loops back to this step, two will be added to X eac time, and X will increase.

```
      ∇TWOS
      [1] X←0
      [2] X←X+2
```

```
[3] X
[4] →2
[5] ∇
```

The right arrow means "go to"; step 4 says "go to step 2"

If you execute this program you will get endless numbers increasing by 2 each time. To stop this program, hit ATTN and after the computer types the line number, you type a right arrow (→). Now you are out of the program.

You need to have some way for this program to stop itself. When *X* reaches a certain number, make the program stop. To do this you must have a conditional loop; in other words, if *X* is less than a certain number, make the loop again (make *X* increase by 2), otherwise, stop.

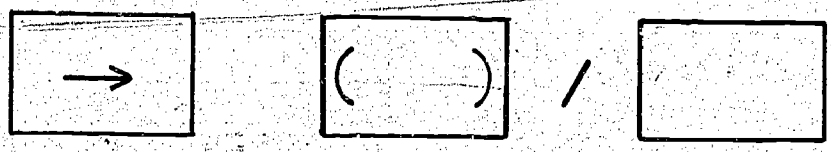
A conditional branch is made of the "go to" arrow, the condition, and the line number.

```
→(X<N)/2
```

N is the number you don't want *X* to be greater than.

If the condition is true, the program will loop, otherwise the program will go to the very next step.

If (*X*<*N*) go to (→) step 2 (2). But the APL order requires that it read: go to (→) if the condition is true (*X*<*N*) step 2 (2).



THREE COMPONENTS OF A CONDITIONAL BRANCH

A = GO TO
B = IF THIS IS TRUE
C = WHERE PROGRAM IS TO GO NEXT

EXAMPLE:

```
      →  (W>4)  /  10
      ↑    ↑       ↑
      A    B       C
```

Your program will interrupt and go immediately to line number 10 if the variable "W" is greater than 4, where it will then do what is required by line 10 and then go on from there. ALL CONDITIONAL BRANCHES MUST CONTAIN THESE THREE ELEMENTS. This useful feature of APL allows you to do something in the program IF a certain condition occurs.

To replace step 4 , reopen the program named *TWOS* :

```
      ∇TWOS
[5]
```

the computer does not respond with line 1 because the first four lines of the program named *TWOS* are already filled up. Instead of entering line five, tell the computer you want to redo line 4:

```
      ∇TWOS
[5] [4]
[4]
```

the computer responds with the line number you wish to rewrite and awaits reentry.

```
      →(X<N)/2
[5] ∇      reenter execution mode.
```

To run this program, assign *N* a number, then type the program name.

```
      N←15
      TWOS
2
4
6
8
10
12
14
```

LIBRARIES AND USER ACCOUNT NUMBERS

The APL system can be broken up into several different parts. When you sign on, you use what is called a user account number. The system has many of these account numbers, all above 1000. The ones numbered below 1000 have a special use: they contain programs that anyone in the system can use. You can't sign on with one of these numbers, but you may "borrow" programs from them to use yourself.

Your account (or library) number -- the one you sign on with -- is divided into one or more workspaces. Each workspace in your library is like a book in a real library. It contains your stored programs (or functions), which are like chapters in the book. You can borrow and use a copy of any book (workspace) in your library. In fact, with the proper information (and permission) you can borrow copies of other people's books or workspaces! But you may only look at and use one workspace at a time. When you have a copy of a workspace (when you have loaded it into your workspace) you may use any of its programs. Also, if you have one workspace already loaded and you load another, the one you previously had is automatically removed.

If you want to list the workspaces contained in the account number you signed on with, type *)LIB* and hit RETURN. The list printed out is a list of the names of the workspaces in that account number. If none are printed, then none have been stored. If you want

to see or use one of these workspaces, type *)LOAD* followed by a space and the name of the workspace you wish to see (remember you can only see one at a time). You must spell the name exactly as it is shown in the *)LIB* listing, no matter how strange it may sound. The computer responds with the time and date that the workspace was stored. You have now loaded the workspace.

```
)LOAD 1628 LEMSIM
SAVED  10.57.39 11/11/70
```

WORKSPACES

Now you would probably like to see what programs have been stored (or saved) in the workspace. To do this, simply type *)FNS* and hit RETURN. The list that follows is the names of all the programs or functions within that particular workspace. They are printed in alphabetical order. To list the variables, type *)VARS* and hit RETURN. Again, the list of variables is alphabetically arranged. You may have to do a little experimenting to find what some of the programs do, but go ahead and try -- you won't hurt anything. By the way, if you have forgotten the name of the workspace you are in, the command *)WSID* followed by RETURN will tell you the name of the workspace. If you have not loaded any workspaces, then you are in a clear workspace, and the computer tells you exactly that by typing *CLEAR WS*.

You may want to create a whole new workspace to save your program or programs in. To do this, first you must be in a clear workspace. Then, without loading anything, type in the programs you want stored in the new workspace in the same way you ordinarily would. When you are through, type *)SAVE* (just like the store command) followed by the name you want the new workspace to have (up to eleven letters or numerals, no spaces allowed) and hit RETURN. Note: This is the name of your workspace, not the name of any program. Sample workspace names are *MATHCLASS*, *GAMES*, *JOHNSWORK*. The computer will respond by typing out the time and date of storage. Your workspace is now saved, along with that program (or function) which you created when you typed the ∇ and the program name. Any time you sign on with your account number, you may type *)LIB* and your workspace will be one of those listed. Remember, however, that the new workspace is stored only in the number you signed on with. Often, when you try to save a new workspace, you will get the message *WS QUOTA USED UP* . This means just that: there is no room left in that number for a new workspace. If this is the case, it is advisable to just use one of the other workspaces in your number. If you need a new workspace urgently, call the computer center downtown; they can give you additional room for workspaces.

PUBLIC LIBRARIES

APL has a group of workspaces with numbers below 1000 that are there for public use. In fact, they are called Public Libraries. *)LIB* followed by the library number, and out comes a list of the workspaces contained in that number. Simply type

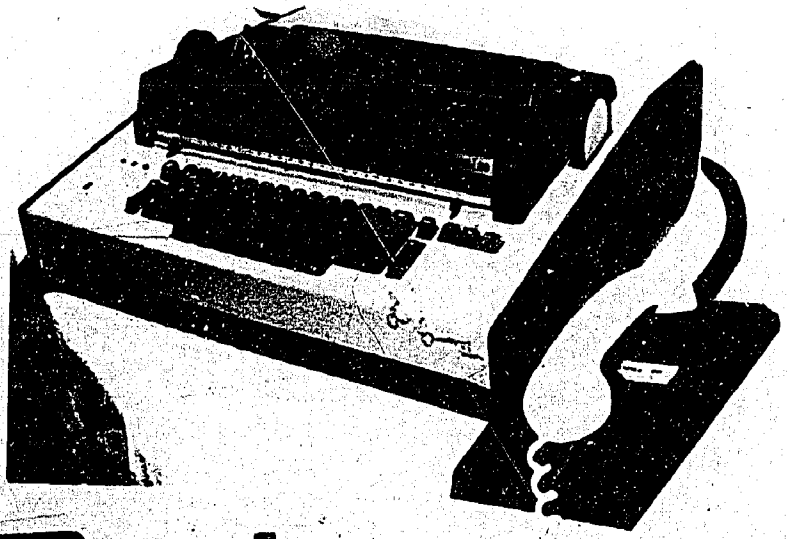
```
      )LIB 1
ADVANCEDEX
APLCOURSE
NEWS
PLOTFORMAT
TYPEDRILL
WSFNS
VIDEO
MULTDRILL
TRYTHIS
LATER
```

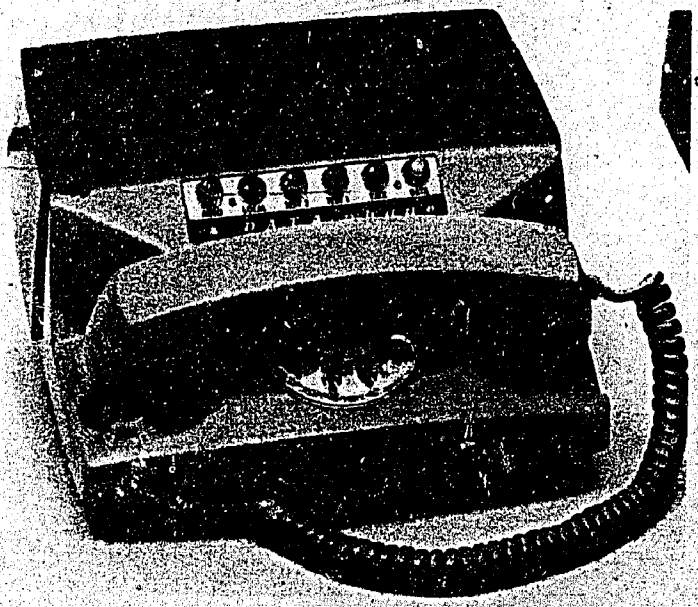
The libraries we currently have are numbered 1 through 5, so give this listing a try. Loading a workspace from a public library is almost exactly like loading one of your own private workspaces. The one difference is that you have to tell what library and what workspace you want when loading from a public library, while with your own workspaces you need only give the name of the workspace. To load a public library workspace, type *)LOAD* followed by the library number and the name of the workspace you want. Note the following example:

```
      )LOAD 1 PLOTFORMAT
SAVED 10.33.29 09/22/68
```

Most of the workspaces in our public libraries will give you some kind of description of their contents if you type out *DESCRIBE* . Even if they don't, you can list the names of the programs they contain by typing *)FNS* . The list that follows (in alphabetical order) is a list of the programs in that workspace. It may require some experimenting to find what some of the programs do, but don't let this frighten you. Experimenting in APL is fine -- especially in public libraries, where nothing can be stored permanently by our regular users.

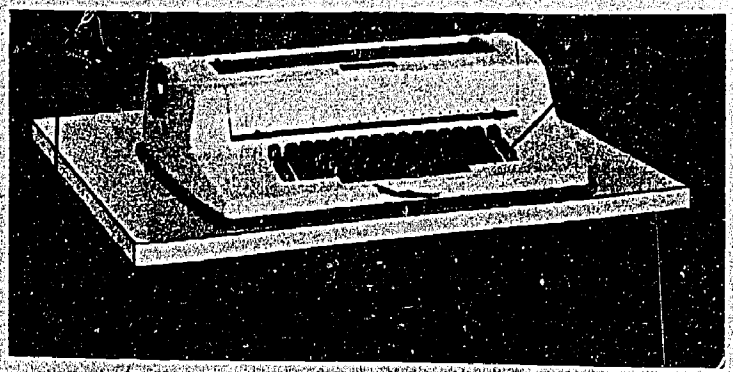
DATEL TERMINAL





TELEPHONE DATASET

IBM 2741 TERMINAL



ERROR REPORTS

CHARACTER ERROR	Improper symbol used -- re-enter line
DEFN ERROR	Incorrect attempt at function definition or improper use of ∇
DOMAIN ERROR	Requested operation cannot be done using the given values
LENGTH ERROR	Vectors are not of the same length -- rework one vector
MESSAGE LOST	The message you were trying to send was never transmitted -- re-send
RESEND	Last line entered was not received by the computer -- re-enter last line
SYMBOL TABLE FULL	Too many variables used -- erase some
SYNTAX ERROR	Incorrect command -- re-enter line with corrections
SYSTEM ERROR	Internal problem -- report these to the Terminal Services Coordinator along with your printout
VALUE ERROR	A variable has not been given any value -- assign a value to the variable
WS FULL	You have no more room in your workspace -- load another one

SYSTEM COMMANDS

)12345	sign on
)OFF	sign off
)CONTINUE	sign off and save your workspace
)LOAD	change to a different workspace
)ERASE	erase a program (function) or a variable
)WSID	name of current workspace
)WSID JOBFOUR	change name of current workspace to *JOBFOUR*
)SAVE	store workspace on disk
)SAVE WSNAME	store a copy of current workspace under a new or different name
)FNS	list the programs (functions) in a workspace
)VARS	list variables in a workspace
)SI	list any halted functions

)LIB	list all named workspaces in your library
)PORTS	list terminals currently signed on the system, by port number and identification
)MSGN	send a one-line message to another port
)OPRN	send a one-line message to the operator at the computer center
)CLEAR	drop the current workspace

I-BEAM FUNCTIONS

The ⌶ (⍳ backspace ⊤ , spelled I-beam) followed by any of these numbers, will give you certain data depending on which number is used:

⌶19	Amount of time (in sixtieths of a second) that the keyboard has been unlocked since sign-on
⌶20	Time of day (in sixtieths of a second)
⌶21	Amount of computer time used since sign-on (in sixtieths of a second)
⌶22	Number of bytes (or characters) remaining for use in your workspace
⌶23	Number of terminals currently in use
⌶24	Time you signed on (in sixtieths of a second)
⌶25	Date, given as pairs of digits for month, day and year: 07 23 70
⌶27	Gives a vector of statement numbers of statements within any programs where execution was suspended for some reason.
⌶26	Gives the first element of the vector developed by I-beam 27.
⌶29	Gives your user account number.

SYMBOL CHART

SYMBOL	NAME	FUNCTION OR MEANING
	TERMINAL ROW ONE	
¨	umlaut or diaeresis	not used
-	negative	monadic - designate a negative number
<	less than	dyadic - comparing numbers
≤	less than or equal to	dyadic - comparing numbers
=	equals	dyadic - comparing numbers
≥	greater than or equal to	dyadic - comparing numbers
>	greater than	dyadic - comparing numbers
≠	not equal	dyadic - comparing numbers
∨	or	dyadic - logical union
∧	and	dyadic - logical intersection
-	minus	monadic-negation dyadic -subtraction
÷	division	monadic-reciprocal dyadic -division
+	plus	monadic-identity dyadic -addition
×	times	monadic-sign dyadic -multiplication
	TERMINAL ROW TWO	
?	roll	monadic-random numbers dyadic -random numbers without replacement
ω	omega	not used
∈	element	dyadic - logical membership
⍴	rho	monadic-dimensions dyadic -reshape
~	not	monadic-logical negation
↑	take	dyadic - use specified elements
↓	drop	dyadic - delete specified elements

⍳	iota	monadic-consecutive integers dyadic -indexing of arrays
○	pi times	monadic-multiples of pi dyadic -trignometric functions
*	exponent	monadic-natural exponent dyadic -power
→	branch	monadic-branch within a program
←	assign	dyadic - assigning values

TERMINAL ROW THREE

⍺	alpha	not used
⌈	ceiling	monadic-rounds up
⌈	maximum	dyadic -takes maximum
⌊	floor	monadic-rounds down
⌊	minimum	dyadic -takes minimum
_	underline	used to underline various letters and characters
∇	del	used to change modes
∆	delta	trace and stop codes
∘	small circle	outer product
'	quote	used only in pairs for literal characters
⍞	quad	numeric input
( )	parenthesis	used in pairs to override order of execution
[ ]	brackets	used in pairs to index

TERMINAL ROW FOUR

⊂	unnamed	not used
⊃	unnamed	not used
∩	unnamed	not used
∪	unnamed	not used
⊥	decode	dyadic - change to base ten
⊤	encode	dyadic - change bases
|	absolute value	monadic-absolute value

|	residue	dyadic -remainder
,	catenation	monadic-make a vector dyadic -combining vectors
;	semicolon	dyadic - separation
.	decimal	place decimal within a number
:	colon	used with lock-words or line labels
\	left slash	dyadic - expand arguments
/	right slash	dyadic - reduce or compress

OVERSTRIKES

SYMBOL	HOW MADE	NAME	FUNCTION OR MEANING
⍟	○ backspace ⋆	logarithm	monadic - natural log of a number dyadic - log of a number, any base
⌽	○ backspace |	reversal	monadic - reverses order
		rotate	dyadic - rotates first elements
⍉	○ backspace \	transpose	monadic - switches dimensions dyadic - changes designated dimensions of a matrix
⍋	∆ backspace |	grade up	monadic - orders in ascending order
⍒	∇ backspace |	grade down	monadic - orders in descending order
⍫	∇ backspace ~	del-tilde	locks functions (USE WITH CARE!)
⍝	∩ backspace ∘	comment or lamp	prevents the execution, allows for comments
⍞	' backspace □	quote-quad	requires literal input

!	' backspace .	factorial combination	monadic - evaluates factorial dyadic-evaluates a combination
⍲	^ backspace ~	nand	dyadic - not and
⍱	v backspace ~	nor	dyadic - not or
⌶	⊥ backspace ⊤	I-beam	monadic - various computer functions

MYSTERII

IF you get no response after entering a command

TRY hitting RETURN

WHY? the computer cannot execute unless you send the command, and it receives the command

IF you get no response and the type element is at the left margin

TRY typing a quote

WHY? you may be in an open quote

IF nothing happens for a long time

TRY entering data, or a command

WHY? you may have forgotten to

IF you execute a program and do not get any response for several minutes

TRY hitting ATTENTION, then a →

WHY? it may be in an endless loop

IF	you dial up and get a busy signal or no answer
TRY	hanging up
WHY?	the system is down
IF	you keep getting []
TRY	entering any number, or →
WHY?	you are in a program requesting numerical input (you'll be out eventually)
IF	you don't understand what you are doing
TRY	doing it anyway
WHY?	you'll never learn unless you do
IF	you think you have damaged the computer because you typed something
TRY	again
WHY?	you haven't hurt anything
IF	the type element is at the left margin (without spacing over) awaiting input
TRY	typing *O* backspace *U* backspace *T* .
WHY?	you are within a program requiring literal or character input: this will get you out
IF	the terminal or telephone itself is actually broken
TRY	calling 659-3381, extension 327, and ask for the Terminal Services Coordinator
WHY?	report all troubles to the Terminal Services Coordinator

We invite your comments, suggestions and criticisms, whether they be bouquets or brickbats! It is planned that this book will be revised within two years, so we would like to make the next edition better and more useful to our readers. Tell us what you like or don't like about this book. If you catch mistakes or errors, please tell us the specific page, paragraph, and line. If you have questions that you would like answered, please be sure to include your name and address. Suggestions may be anonymous if you choose.

Address your comments to:

Terminal Services Coordinator
Information Processing System
Atlanta Public Schools
218 Pryor St., S.W.
Atlanta, Georgia 30303

Thank you!

LIST OF REFERENCE PUBLICATIONS:

APL\360 GENERAL INFORMATION MANUAL. SECOND EDITION. GH-20-0850-1. IBM CORP. DECEMBER, 1970.

APL\360 PRIMER. GH20-0689-0. IBM CORP. 1969.

APL\360 USER'S MANUAL. FIRST EDITION. IBM CORP. DECEMBER, 1970.

APL\360 REFERENCE MANUAL. SANDRA PAKIN. PUBL. 17-1. SCIENCE RESEARCH ASSOCIATES. 1968.

APL PROGRAMMING AND COMPUTER TECHNIQUES. HARRY KATZAN, JR. VAN NOSTRAND - REINHOLD CO. 1969.

AN INTRODUCTION TO APL\360 WITH SOME STATISTICAL APPLICATIONS. K.W. SMILLIE. PUBL. NO. 19. UNIVERSITY OF ALBERTA, CANADA. JANUARY, 1970.

This entire book was composed, edited and typeset using ATS/360, another terminal service of the Atlanta Public Schools.